(12) # EUROPEAN PATENT APPLICATION

(72) Inventor : **Asakawa, Takeo**
c/o Fujitsu Limited, 1015, Kamikodanaka
Nakahara-ku, Kawasaki-shi, Kanagawa 211
(JP)
Inventor : **Inoue, Aiichiro**
c/o Fujitsu Limited, 1015, Kamikodanaka
Nakahara-ku, Kawasaki-shi, Kanagawa 211
(JP)

(74) Representative : **Billington, Lawrence Emlyn et**
al
HASELTINE LAKE & CO Hazlitt House 28
Southampton Buildings Chancery Lane
London WC2A 1AT (GB)

(54) Pipeline operation control method and system.

(57)   A pipeline operation control method and system in which, in the execution of continuous first and second instructions, to avoid a pipeline disturbance, the second instruction is shifted from a calculating state to a special state as a final state that does not conflict with the final state of the first instruction when the first instruction has two or more calculating states immediately before the final state, and when the special state can be inserted as the final state of the second instruction and the calculating state of the second instruction is carried out within a single machine cycle, whereby the final state of the first instruction and the special state of the second instruction are carried out at the same time.

Fig. 7

EP 0 503 968 A2

The present invention relates to a pipeline operation control method and system, and more particularly relates to a pipeline operation control method and system for preventing a pipeline disturbance so as to speed up the execution of pipeline instructions.

In a pipeline process, one instruction process is divided into a plurality of processes referred to as states, and plural instructions are executed in parallel formation.

When at least two of the continuous states in a first instruction are the same, these states are called inter locked states. In the inter locked states, the same hardware is used. Therefore, with respect to a second instruction to be executed parallel with the first instruction, the states processed at the same time when the inter locked states are processed must be different from the inter locked states of the first instruction in order to prevent the use of the same hardware in both the first instruction and the second instruction. Because of this, as later described in more detail with reference to the drawings, conventionally, the processing of the instructions followed by the first instruction is delayed. This delay is called a pipeline disturbance.

As a result, conventionally, when there are inter locked states in one instruction, instructions followed by the first instruction are necessarily delayed.

The present invention is provided in view of the above problem, and has an object to provide a pipeline operation control system and method in which the processing time is made short without delay owing to the pipeline disturbance.

To attain the above object, there is provided, according to the present invention, a pipeline operation control method for executing a plurality of instructions by a pipeline process. Each of the instructions has a sequence of a plurality of states including a final state and at least one calculating state immediately before the final state. In a pipeline process of continuous first and second instructions, the method comprises the steps of: a first step for determining whether or not the first instruction has two or more calculating states immediately before the final state; a second step for determining whether or not a special state, which does not conflict with the final state of the first instruction, can be inserted as the final state of the second instruction, and whether or not the calculating state of the second instruction is carried out within a single machine cycle; and shifting the process of the second instruction from the calculating state to the special state as the final state, when the first step determines that the first instruction has two or more calculating states immediately before the final state, and when the second step determines that the special state can be inserted as the final state of the second instruction and that the calculating state of the second instruction is carried out within a single machine cycle. Whereby the final state of the first instruction and the special state of the second instruc-

tion are carried out at the same time.

According to an aspect of the present invention, after the execution of the first and the second instructions simultaneously, the method further comprising the step of storing information necessary to complete the execution of the first and the second instructions, and to control the order of the sequence of the instructions followd by the first and the second instructions.

The information necessary to complete the execution of the first and the second instructions, and to control the order of the sequence of the instructions followd by the first and the second instructions is the location of the instruction that has caused the interrupt and the cause of the interrupt.

When an interrupt is generated immediately after completion of the execution of the first instruction, the information necessary to complete the execution of the first and the second instructions, and to control the order of the sequence of the instructions followd by the first and the second instructions is information necessary for continuing the pipeline process after execution of the first instruction.

After the execution of the first and the second instructions, when an interrupt is not generated immediately after completion of the execution of the first instruction, the information necessary to complete the execution of the first and the second instructions, and to control the order of the sequence of the instructions followd by the first and the second instructions is the information necessary for continuing the pipeline process after the execution of the second instruction.

The information necessary to designate a location of an interrupt is information concerning the cause of a program interrupt.

The information concerning the cause of a program interrupt is an access exception detected in an operand access of an instruction.

Alternatively, the information concerning the cause of a program interrupt is information for designating a program event record detected during the execution of the first or the second instruction.

Still alternatively, the information concerning the cause of a program interrupt is control data necessary to cause a program interrupt.

The information necessary to complete the execution of the first and the second instructions, and to control the order of the sequence of the instructions followd by the first and the second instructions is the address of an instruction that caused an interrupt, the address being stored in a PSW address register.

According to another aspect of the present invention, there is provided a pipeline operation control system for executing a plurality of instructions by a pipeline process as described above. The system comprises a program status word instruction address register for storing an address of the second instruction after the execution of the second instruction

unless an interrupt is generated after the execution of the first instruction, and for storing an address of the first instruction when an interrupt is generated after the execution of the first instruction.

The system further comprises a first register for storing an instruction length code of the first instruction when the first instruction has at least two calculating states immediately before the final state, or for storing an instruction length code of the second instruction when the second instruction does not have the special state, a second register for storing an instruction length of the second instruction only when the second instruction has the special state, a first adding unit for adding the contents in the first register and the contents in the second register, a selecting unit for selecting the output of the first register only when the second instruction does not have the special state, and for selecting the output of the adding unit only when the second instruction has the special state; and a second adding unit for adding the output of the selecting unit and the output of the program status word instruction address register.

The selecting unit has an inhibiting function for inhibiting the selection of the output of the adding unit when an interrupt is detected in the execution of the first instruction.

The system further comprises a program interrupt code register for storing the cause of an interrupt.

The system further comprises a first interrupt content storing register, a second interrupt content storing register, a register output selector for selecting the contents in the first interrupt content storing register or the contents in the second interrupt storing register to be input to the program interrupt code register, the first interrupt content storing register registering the contents of an interrupt of the first instruction when the first instruction has at least two calculating states immediately before the final state or registering the contents of an interrupt of the second instruction when the second instruction does not have the special state, and the second interrupt content storing register registering the contents of an interrupt of the second instruction only when the second instruction has the special state, the register output selector selecting the output of the second interrupt content storing register unless the first interrupt content storing register stores the contents of the interrupt of the second instruction.

The above object and features of the present invention will be more apparent from the following description of the preferred embodiments with reference to the drawings, wherein:

Fig. 1 is a diagram showing a conventional example of a pipeline process;

Fig. 2 is a block diagram showing an example of a conventional pipeline system;

Fig. 3 is a diagram for explaining a pipeline interlock in a conventional pipeline system;

Fig. 4 is a diagram showing an improvement of a conventional pipeline process;

Fig. 5 is an explanatory diagram of a problem in the conventional system;

Figs. 6A and 6B are a flowchart explaining the operation of a pipeline operation control method according to an embodiment of the present invention;

Fig. 7 is a diagram showing the shift of the states in the pipeline operation control method according to an embodiment of the present invention;

Fig. 8 is a diagram explaining the operation of a pipeline operation of a floating point adding instruction and a store instruction when there is a special state, according to an embodiment of the present invention;

Fig. 9 is a diagram explaining the operation of a store instruction when there is not a special state, according to an embodiment of the present invention;

Fig. 10 is a diagram explaining the operation of a pipeline operation of an add instruction and a store instruction when there is not a special state, according to an embodiment of the present invention;

Fig. 11 is a block diagram showing tag registers used in an embodiment of the present invention;

Fig. 12 is a block diagram showing an example of a PSWIAR updating circuit according to an embodiment of the present invention;

Fig. 13 is a flowchart explaining the the operation of a program status word instruction address register in the pipeline operation control method according to an embodiment of the present invention;

Fig. 14 is a block diagram showing excetion code registers used in an embodiment of the present invention;

Fig. 15 is a block diagram showing an example of a program interrupt code register updating circuit according to an embodiment of the present invention; and

Fig. 16 is a flowchart explaining the operation of a program interrupt code register in the pipeline operation control method according to an embodiment of the present invention.

For better understanding of the present invention, conventional techniques are first described.

Figure 1 shows an example of a conventional pipeline operation control. In the figure, each instruction has six states D, A, T, B, E, and W in this example. The process of one instruction is completed by sequentially processing these states.

In the D state, the instruction is decoded; in the A state, the operand address or the target instruction address of the decoded instruction is calculated; in the T state, a search in a TLB (Translation Lookaside Buffer) is carried out to provide an address for acces-

sing a cache memory; in the B state, the cache (or buffer) is read; in the E state, an arithmetic operation is carried out; and in the W state, the result of the arithmetic operation is stored in a register.

First, an instruction 1 is started, and when the instruction 1 enters the state A, the D state of the instruction 2 is started simultaneously. Further, when the instruction A enters the state A, the D state of the instruction 3 is started simultaneously. This follows in a similar way so that a plurality of instructions (in the figure up to the instruction 6) are sequentially executed.

Thus, a plurality of instructions are executed parallel to each other. This execution is similar to a flow of pipelines. Therefore, the operation is referred to as a pipeline operation.

When the state of the instruction enters the state A, the state of the next instruction 2 enters the state D. When the state of the instruction 2 enters the state A, the state of the next instruction 3 enters the state D. Thus, by using the same hardware, a pipeline operation is sequentially executed. To be able to carry out such a pipeline operation, it is necessary not to set the same states of plural instructions at the same time. For example, with respect to a time T shown in Fig, 1, the instruction 1 is in the E state, the instruction 2 is in the B state, the instruction 3 is in the T state, the instruction 4 is in the A state, and the instruction 5 is in the D state. Therefore, at the same time, the same states do not overlap. This is because the same states uses the same hardware. If one tried to process the same states of two instructions at the same time, the hardware would conflict.

The information such as an operand code used in each state of each instruction is stored in a register called a pipeline tag. The pipeline tags are constituted by a shift register. In accordance with the development of the pipeline process, the contents in the pipeline tag of each instruction is shifted to the adjacent tag. The time necessary for one state process is called one machine cycle. When the pipeline process is ideally completed, N instructions seem to be executed within N machine cycles so that the process is executed at a high speed.

Figure 2 is a block diagram showing an example of a conventional pipeline system for executing one instruction. In the figure, the hardware constructions are shown to correspond to respective states. In the D state, an opcode in the instruction is decoded to obtain a base register number B and an index register number X in the operand. By the obtained base register number B and the index register number X, a register stack (including a general purpose register GR and a floating point register FR) is accessed to read out address calculating data set in a base register 2 and an index register 3.

In the next state A, the outputs of the registers 2 and 3 are added by an adder 4, and the result is set

as a valid address in a register 5. In the T state, the valid address set in the register 5 is input to a TLB (Translation Lookaside Buffer) 6. The transformed real address is set from the TLB 6 to a register 7.

In the next B state, the real address set in the register 7 is read and input as an address to a LBS 8 (Local Buffer Storage for storing a copy of part of main storage data so as to enable high speed accessing), and corresponding data is read from the LBS 8 and set in a register 9.

In the next E state, an arithmetic operation on the contents of the register 9 is executed by an arithmetic operation unit 11, and the result of the arithmetic operation is set in a register 12. In the next W state, the result data of the arithmetic operation read from the register 12 is written in a register stack 13. In the next instruction, the contents of the register stack 13 may be set in the B state in a register 10 through a register operand bus 14, and may be sent to the calculating unit 11 in the state E of the next instruction.

In the data processing unit for carrying out a pipeline process, as a factor to prevent an ideal operation of pipeline processing, there are pipeline interlocks. As a typical example of a pipeline inter locks, there is an E state interlock in which a calculation operation is not finished within one cycle. For example, in a floating-point instruction, a fraction alighment operation requires one cycle and an adding operation requires one cycle so that a total of two cycles are required. Accordingly, it is impossible to execute such an instruction as the floating-point instruction by one cycle.

Figure 3 is an explanatory diagram of a pipeline interlock. In the figure, assume that the E state in each of the instructions 1 to 6 requires two cycles. Therefore, the E state in the instruction 1 requires two cycles so that the B state in the next instruction 2 cannot be shifted to the E state at the time of the second E state of the instruction 1 in order to avoid the hardware conflict, resulting in the B state in the instruction 2 using two cycles. Therefore, even though the instruction 1 is finished by seven cycles, on and after the instruction 2, the length of each instruction is sequentially increased by one cycle. As a result, the D state in the instruction 5 requires two cycles. Thus, the start of the instruction 6 is delayed by one cycle in comparison with the case of Fig. 1. The reason for this is that the influence of the fact that the E state in the instruction 1 requires two cycles extends to the instruction 5 so that the D state requires two cycles.

Figure 4 is a diagram of another conventional method of an improved pipeline process. To eliminate the inconvenience as described with reference to Fig. 3, the method shown in Fig. 4 makes the E state, that is an arithmetic operation state, into a pipeline so as to divide it into an E state and a V state. Namely, in Fig, 4, each of the interlocked E states in all instructions as shown in Fig. 3 is divided into the E state and

the V state to carry out a pipeline process. In this case, the W state in one instruction does not conflict with the V state in the next instruction so that a smooth processing similar to the process shown in Fig. 1 becomes possible so that the D state in each instruction is sequentially executed in a one cycle delay.

When an instruction with a V state and an instruction without a V state are continuously executed, however, the instructions are also delayed.

Figure 5 is a diagram explaining the problem in the conventional method shown in Fig. 4. Assume that the instruction 1 is with a V state, the instruction 2 is without a V state, and the instructions on and after the instruction 3 are with V states respectively. In the conventional art, the processes of all of the instructions must be finished at the W tates. Therefore, if the instruction 2 does not have the V state, the instruction 2 must be interlocked before the preceding instruction 1 enters the W state in order to prevent that the second instruction 2 goes beyond the W state of the preceding instruction 1. Generally, during the E state, an arithmetic unit that is driven with a fixed clock signal; by constrast, during the B state, the arithmetic unit is not yet operated but data is read from a chache or an operand is read from a register and the read data or the operand is sent to the arithmetic unit. Therefore, the B state can freely be interlocked. Because of this reason, the instruction 2 is interlocked by using the two cycles of the B state. Thus, during a period when the instruction 1 is executing the E state and the V state, the instruction 2 must be kept in the B state. During the two cycles of the B states in the instruction 2, the states of the following instructions cannot be shifted to the next state so that each instruction is kept in the previous state for two cycles.

As a result, the instruction 5 is kept in the D state for two cycles, and the instruction 6 enters the D state after the D state of the instruction 5 is finished. As a result, a delay of one cycle is generated between the instruction 5 and the instruction 6.

The present invention removes the above-described problems in the conventional methods.

Figures 6A and 6B are a flowchart explaining the operation of a pipeline operation control method according to an embodiment of the present invention.

In Fig. 6A, at step S61, during the process of the first instruction, it is determined whether or not the process of the first instruction enters from the E state to the V state. If so, at step S 62, it is determined whether or not a special state S, which does not conflict with the W state in the first state, can be inserted as the last state of the second instruction. If it is possible, at step S 63, the last state of the second instruction is shifted from the E state to the S state. Then, at step S64, the W state of the first instruction and the S state of the second instruction are executed at the same time. Then, at step S65, it is determined whether or not an interrupt is generated because of

the execution of the first instruction. If it is yes, then at step S66, the instruction complete tag for the first instruction is retained. Here, the instruction comlete tag is an instruction length code (ILC) and an exception valid XV. If the interrupt is not generated by the first instruction, then at step S67 the instruction complete tag for the second instruction is retained. Then, at step S68, the contents of the program status word instruction address register (PSWIAR) are obtained from the instruction complete tag for the first or the second instruction.

At step S61 or S62, if the answer is no, then from steps S690 to S694 as shown in Fig. 6B, the conventional pipeline process as shown in Fig. 4 or Fig, 5 is carried out. Namely, at step S690, the last two states of the second instruction are made as the V state and the W state; at step S691, the W state of the first instruction and the V state of the second instruction are executed at the same time; at step S692, the contents of the PSWIAR of the first instruction are obtained; at step S693, the W state of the second instruction is executed; and at step S694, the contents of the PSWIAR of the second instruction are obtained.

According to the embodiment of the present invention, when the special state S can be set in the second instruction so that the first and the second instruction are completed at the same time; the second instruction and those after the second instruction are not delayed so that the process can be carried out at a high speed.

The S state is the final state of the second instruction for finishing a pipeline processing of the second instruction when the second instruction can be shifted into the S state.

Figure 7 is a diagram explaining a pipeline operation according to an embodiment of the present invention. In the figure, the second instruction 2, which is referred to as an S cycle instruction, follows the instruction 1. In the last two states of the S cycle instruction, the pipeline processing state is shifted from the E state to the S state. In other words, the S cycle instruction does not include the V state after the E state. Therefore, the pipeline finishing process, i.e., the W state, of the instruction 1 immediately before the S state instruction 2 and the S state of the S cycle instruction 2 are carried out simultaneously. Note that the S state is selected in such a way that it does not conflict with the W state of the first instruction. Namely, the hardware used in the W state of the first instruction is different from the hardware used in the S state of the second instruction. Further, the E state in the S cycle instruction is selected in such a way that it does not conflict with the V state of the first instruction. As a result, The B state in the S state instruction 2 is within a single cycle, and thus, the D state of each instruction is not forced to wait so that the time is shorter by one cycle in comparison with the conventional case shown in Fig. 5. In Fig. 7, at the timing of the W

state of the instruction 1, pipeline finishing processes of two instructions are carried out simultaneously by the information of the W state of the instruction 1 and the information of the S state of the S cycle instruction.

Fig. 8 is a diagram explaining the operation of a pipeline operation of a floating point adding instruction (AD) and a store instruction when there is a special state, according to an embodiment of the present invention. In Fig. 8, the store instruction includes the S state as its final state.

When the result of the floating point adding instruction (AD) is not used in the store instruction, the result of a previously executed instruction, which is different from the floating point adding instruction (AD), is stored by the store instruction. Namely, the contents in a register 13, which have been stored in accordance with the previously executed other than the floating point adding instruction AD, are read at the B state in the store instruction and set in a register 10, and the selector 23 selects the output of the register 10 so that the contents of the register 10 are set through a selector 23 and a cache storage bus (CSB) 21 to a store data register (STDR) 22.

By contrast, when the result of the floating point adding instruction is to be stored by the store instruction, the selector 23 selects the output of the register 12, in which the contents of the calculation effected during the E state of the AD instruction are stored, and the contents in the register 12 are stored through selector 23 and the CSB 21 to the store data register (STDR) 22. Note that the storing operation into the store data register (STDR) 22 is carried out immediately in the S state.

As a result, when the result of the AD instruction is to be stored, the data in the register 12 is set through the selector 23 to the STDR 22. As a result, the W state of the AD instruction and the S state of the store instruction are completed at the same time. Since the B state in the store instruction is only one cycle, there is no delay of two cycles at the start of the instruction 6. In Fig. 8, the next instruction followed by the store instruction is an instruction without a V state. Therefore, since the store instruction does not have the state B over the two cycles, there is no time loss in the third instruction and those after the third instruction.

It should be noted, however, that the S state is not always inserted in the instruction, but it is also possible to execute an instruction without the S state.

Figure 9 is a diagram for explaining a store instruction when it does not include the S state in the last state according to an embodiment of the present invention. In this case, the operation is the same as the conventional store operation. Namely, the contents in a register stack 13 are read and set in the B state. The output of the register 10 is set through the CSB 21 in the store data register (STDR) 22 during the E state. Then, from the STDR 22, the data is

stored in a cache or a main storage. After setting the data in the STDR 22, the pipeline process for the store instruction is completed.

Figure 10 is a diagram explaining the operation of an add instruction and a store instruction when the store instruction does not include the S state in the last state according to an embodiment of the present invention.

The figure shows the case in which a result of an A instruction (fixed point add instruction) is stored. In this case, the operation is the same as the conventional operation. Namely, when the add instruction and the store instruction are not related to each other, according to the add instruction, the result of the calculation in the arithmetic operation unit 11 is set in a register 12, and the contents of the register 12 are set in a register stack 13; and according to the store instruction, the data set in the register stack 13, which were previously stored by the an instruction other than the above mentioned A instruction, are read during the B state from the register stack 13, and are set in the register 10. The contents of the register 10 are selected by a selector 23 during the E state in the store instruction and set in the STDR 22. By contrast, when the result of the A instruction is to be stored in the STDR 22 by the store instruction, the result of the calculation during the E state by the arithmetic operation unit 11 is set in the register 12 during the E state in the A instruction, and the contents of the register 12 are selected instead of the register 10 by the selector 23 during the E state of the store instruction to be output to the CSB 21 and set in the STDR 22 during the E state in the store instruction.

As a result, while two cycles of instructions are necessary in the sequence in which the contents of the register stack 13 are read and stored, it is possible, by directly writing the contents of the register 12 into the STDR 22, to execute the store instruction by one cycle when the E state interlock is not present in the A instruction. This is the same as in the conventional method.

Since the first instruction and the second instruction are completed at the same time as the second instruction includes the S state at its last stage, the program status word instruction address register must be designed to store the correct address of the program regardless of whether the instruction with the S state or the instruction without the S state is finished.

Figure 11 is a block diagram showing tag registers for one instruction. In the figure, during the D state, an operand code is read from an instruction register. Based on the operand code, an instruction length code ILC is generated by an ILC generating circuit 112. The instruction length code ILC is sequentially shifted in an A-ILC register 113 during the A state, a T-ILC register 114 during the T state, a B-ILC register 115 during the B state, and an E ILC register during the E state, respectively. After the E state,

when the state of the instruction is changed from the E state to the W state, a selector 30 selects, during the W state, the ILC in the E-ILC register to be stored in a W-ILC register 31; when the state is changed from the E state to the V state, the ILC is stored in a V-ILC register 117 during the V state, and the selector 30 selects the ILC in the V-ILC register 117 to be stored in the W-ILC 31 during the W state; and when the state is changed from the E state to the S state, the ILC is shifted from the E-ILC register 116 to an S-ILC register 32. These registers A-ILC, T-ILC, B-ILC, E-ILC, V-ILC, S-ILC, and W-ILC are referred to as tag registers.

Figure 12 is a block diagram showing a circuit for updating the program status word register instruction address register (PSWIAR). In the figure, 30 is the selector, 31 is the W-ILC register or in other words a W state instruction length register, 32 is the S-ILC register or in other words an S state instruction length register, 33 is an adder, 34 is a selector, 35 is an adder, and 36 is the PSWIAR.

Each of the instruction length codes ILC is retained as a tag of the W state instruction that has the W state as its last state, i.e., a tag W-ILC, or a S state instruction which has the S state as its last state, i.e., a tag S-ILC. A W-ILC is set in the register 31 by setting a V-ILC (instruction length code of the instruction in the V state) or a E-ILC (instruction length code of the instruction in the E state) through the selector 30. On the other hand, A S-ILC is set in the register 32 by the E-ILC. Here, the V-ILC represents the instruction length code of an instruction being executed in the V state, and the E-ILC represents the instruction length code of an instruction being executed in the E state.

The adder 33 adds the W-ILC and the S-ILC. The selector 34 selects either one of the W-ILC and the output of the adder 33. The output of the selector 34 is added to the output of the PSWIAR 36 in an adder 35 to update the PSWIAR 36. The output of the PSWIAR 36 is input to the register 10 (see Fig. 8).

The PSWIAR 36 is updated in the final state of the instruction process, namely the last W state or S state. Therefore, the contents of the PSWIAR 36 represent the address of the last instruction that has been processed. When an interrupt is generated, the PAWIAR 36 is used to indicate the instruction that caused the interrupt. To this end, the contents of the PSWIAR 36 are stored during the interrupt process by hardware from the register 10 to a designated main storage area as an old PSW through the CSB 21, and is referred to by an interrupt process program.

Here, The selecting condition in the selector 34 in particular is explained. The condition for the W-ILC to be selected in the selector 34 is when a program interrupt is detected in the W state of an instruction, namely in an instruction having the W state at its last state. The example in Fig. 7 is the case when a program interrupt is detected in the instruction 1. When the PSWIAR 36 indicates that the instruction 1 is com-

pleted, and when an interrupt is detected during the execution of the instruction 1, then the S state instruction (the store instruction in the above example) must be inhibited to be executed. The inhibition of the store instruction is carried out by inhibiting the storing operation of data into the store data register (STDR) 22 in Fig. 8 and by making the selector 34 to select the output of the W-ILC 31 in Fig. 11.

When an interrupt is not generated after the execution of the instruction 1, the selector 34 selects the output of the adder 33 so that the instruction length codes of two instructions, i.e., the first instruction and the second instruction, are added to the contents of the PSWIAR 36. Therefore, the PSWIAR 36 indicates that the instruction up to the S state instruction has been completed at that time.

Figure 13 is a flowchart explaining the operation of the program status word instruction address register shown in Fig. 12 in the pipeline operation control method according to an embodiment of the present invention.

In Fig. 13, at step S71, when the last two states of the first instruction changes from the V state to the W state, or when the last two states of the second instruction change from the E state to the S state, then at step S 72, the selector 30 selects the V-ILC (the instruction length code of the instruction in the V state) of the first instruction to set in the W-ILC 31, and the E-ILC (the instruction length code of the instruction in the E state) of the second instruction is set in the S-ILC 32. Then, at step S73, the adder 33 adds the W-ILC and the S-ILC. At step S74, it is determined whether or not the interrupt is caused because of the first instruction. If t is yes, then at step S 75, the selector 34 select the W-ILC register 31 so that the adder 35 outputs PSWIAR + W-ILC as a new PSWIAR. At step S74, it the interruption is not caused by the first instruction, then at step S 79, the selector selects the output of the adder 33 so that the adder 35 outputs PSWIAR + S-ILC + W-ILC as a new PSWIAR.

At step 71, when the last two states of the instruction do not change from the V state to the W state, then at step S76, it is determined whether or not the last two states of the instruction change from the E state to the W state. If so, at step S77, the E-ILC is set in the W-ILC 31. Then, at step S78, the W-ILC is added to the contents of the PSWIAR 36 and the added result is stored in the PSWIAR 36. In this case, there is no S-state instruction.

Figure. 14 is a block diagram showing exception code registers used in an embodiment of the present invention. in the figure, 141 is a program interruption detecting circuit for detecting a program interruption. When the circuit 141 detects a program interruption during the E state of an instruction, it generates an exception valid code EXV and an exception code EXC. The exception valid code XV is a code for representing the presence of the interruption condition.

The exception code XC is a code for identify the cause of the interruption. Here, the character, for example E, on the head of the EXV and EXC means that the interruption is detected during the E state.

When the state of the instruction is changed from the E state to the W state, a selector 146 selects the EXV which is then stored in a WXV register 146 for storing the exception valid code WXC, and a selector 40 selects the EXC which is then stored in a WXC register 41. When the state of the instruction is changed from the E state to the V state, the EXV is shifted through a VXV register 142 and the selector 146 to the WXV register 143, and the EXC is shifted through a VXC register 145 and the selector 40 and the selector 40 to the WXC register 41. When the state of the instruction is changed from the E state to the S state, the EXV is stored in a SXV register 144.

Figure 15 is a block diagram showing an example of a program interrupt code register updating circuit according to an embodiment of the present invention.

When an interrupt is generated, the cause of the interrupt must also be retained. Similar to the instruction length code ILC, exception code XC is retained as a tag VXC or EXC of the V state instruction or the E state instruction. Either one of the VXC and EXC is selected by a selector 40 and is set in a WXC register 41. The EXC is set in the SXC register 42.

The WXC or the SXC is selected by a selector 43, and is set in a program interrupt code register (PGMIC) 44. The PGMIC 44 is stored in a designated main memory area during an interrupt process by hardware. Here, the selector 43 select the WXC when an interrupt condition is present in the W state, and selects the SXC when the interrupt condition is not present in the W state but present in the S state to update the PGMIC 44, When both the W and S states do not have any interrupt condition, the PGMIC 44 is not updated.

Figure 16 is a flowchart explaining the operation of the program interrupt code register shown in Fig. 15 in the pipeline operation control method according to an embodiment of the present invention. In the figure, at step S 160, it is determined whether the state of the first instruction changes from the V state to the W state and the state of the second instruction changes from the E state to the S state. If so, at step S161, the selector 40 selects the VXC to be stored in the WXC register 41, and EXC is directly stored in the SXC register 42. Then, at step S162, it is determined whether or not the interruption is because of the first instruction. If so, at step S 163, the selector 43 selects the WXC to be stored in the PGMIC 44. At step S162, if the interruption is not caused by the first instruction, then at step S 164, it is determined whether or not the second instruction is the cause of the interruption. If so , at step S165, the selector 43 selects the SXC to be stored in the PGMIC 44. At step S166, if the answer is no, then at step S167, it is determined whether or

not the interruption is caused by a W state instruction. If so, at step S168, the selector 43 selects the WXC to be stored in the PGMIC 44.

In the above description, the embodiment of the present invention is described in detail by using the store instruction, however, the present invention is not restricted to this. For all cases in which hardware resources do not conflict except for the operation of the PSWIAR and so forth in the W state, the present invention can be applied. As an example of such cases, for example, a combination of a floating point instruction and a condition branching instruction can be considered.

In this case, by shifting the condition branching instruction from the E state to the S state, the condition branching instruction can be finished simultaneously with the preceding instruction as shown in Fig. 5 to improve the performance in the speed of the operation. In this case, the difference from the previously explained store instruction is that, in the S state of the condition branching instruction, whether or not the PSWIAR should be replaced by a target address of a branch which is retained as a pipeline tag is controlled by the exception event of the floating point instruction, and depends on whether or not the condition for branching of the condition branch instruction is established.

As described above in detail, according to the present invention, a pipeline calculation control method and system in which the delays of instructions are eliminated and a high speed operation is realized can be provided by additionally providing a special state that can be set simultaneously with the state of the normal pipeline process, and when an instruction can be shifted to the special state by executing the pipeline finishing processes of two instructions simultaneously.

Claims

1. A pipeline operation control method for executing a plurality of instructions by a pipeline process, each of said instructions having a sequence of a plurality of states including a final state and at least one calculating state before the final state, wherein in a pipeline process of continuous first and second instructions, the method comprising the steps of:

   a first step for determining whether or not said first instruction has two or more calculating states before the final state;

   a second step for determining whether or not a special state can be inserted as the final state of said second instruction, and whether or not the calculating state of said second instruction is carried out within a single machine cycle; and

   shifting the process of said second instruc-

tion from said calculating state to said special state as the final state when said first step determines that said first instruction has two or more calculating states, and when said second step determines that said special state can be inserted as the final state of said second instruction and that the calculating state of said second instruction is carried out within a single machine cycle;

the final state of said first instruction and the special state of said second instruction being carried out at the same time.

2. A pipeline operation control method for executing a plurality of instructions by a pipeline process, each of said instructions having a sequence of a plurality of states including a final state and at least one calculating state immediately before the final state, wherein in a pipeline process of continuous first and second instructions, the method comprising the steps of:

a first step for determining whether or not said first instruction has two or more calculating states immediately before the final state;

a second step for determining whether or not a special state, which does not conflict with the final state of said first instruction, can be inserted as the final state of said second instruction, and whether or not the calculating state of said second instruction is carried out within a single machine cycle; and

shifting the process of said second instruction from said calculating state to said special state as the final state when said first step determines that said first instruction has two or more calculating states immediately before the final state, and when said second step determines that said special state can be inserted as the final state of said second instruction and that the calculating state of said second instruction is carried out within a single machine cycle;

the final state of said first instruction and the special state of said second instruction being carried out at the same time.

3. A pipeline operation control method as claimed in claim 2, wherein after the execution of said first and the second instructions simultaneously, the method further comprising the step of storing information necessary to complete the execution of the first and the second instructions, and to control the order of the sequence of the instructions followed by said first and second instructions.

4. A pipeline operation control method as claimed in claim 3, wherein said information necessary to complete the execution of the first and the second instructions, and to control the order of the sequ-

ence of the instructions followed by said first and second instructions is the location of the instruction that has caused the interrupt and a cause of the interrupt.

5. A pipeline operation control method as claimed in claim 3, wherein when an interrupt is generated immediately after the completion of the execution of said first instruction, said information necessary to complete the execution of the first and the second instructions, and to control the order of the sequence of the instructions followed by said first and second instructions is information necessary for continuing the pipeline process after the execution of said first instruction.

6. A pipeline operation control method as claimed in claim 3, wherein after the execution of said first and the second instructions, when an interrupt is not generated immediately after the completion of the execution of said first instruction, said information necessary to complete the execution of the first and the second instructions, and to control the order of the sequence of the instructions followed by said first and second instructions is the information necessary for continuing the pipeline process after the execution of said second instruction.

7. A pipeline operation control method as claimed in claim 4, wherein said information necessary to designate a location of an interrupt is information for causing a program interrupt.

8. A pipeline operation control method as claimed in claim 7, wherein said information for causing a program interrupt is an access exception detected in an operand access of an instruction.

9. A pipeline operation control method as claimed in claim 7, wherein said information for causing a program interrupt is information for designating a program event record detected during the execution of said first or said second instruction.

10. A pipeline operation control method as claimed in claim 7, wherein said information for causing a program interrupt is control data necessary to cause a program interrupt.

11. A pipeline operation control method as claimed in claim 3, wherein said information necessary to complete the execution of the first and the second instructions, and to control the order of the sequence of the instructions followed by said first and second instructions is the address of an instruction that caused an interrupt, said address being stored in a PSW address register.

12. A pipeline operation control system for executing a plurality of instructions by a pipeline process, each of said instructions having a sequence of a plurality of states including a final state and at least one calculating state before the final state; when a first instruction includes at least two calculating states before the final state, a second instruction followed by said first instruction includes a special state as the final state of said second instruction; said special state does not conflict with the final state of said first instruction and the final state of said first instruction and the special state of said second instruction are carried out at the same time, said system comprising:

a program status word instruction address register (PSWIAR) for storing an address of said second instruction after the execution of said second instruction unless an interrupt is generated after the execution of said first instruction, and for storing an address of said first instruction when an interrupt is generated after the execution of said first instruction.

13. A pipeline operation control system as claimed in claim 12, further comprising:

a first register (W-ILC) for storing an instruction length code of said first instruction when said first instruction has at least two calculating states immediately before the final state, or for storing an instruction length code of said second instruction when said second instruction does not have said special state;

a second register (S-ILC) for storing an instruction length of said second instruction only when said second instruction has said special state;

a first adding means (33) for adding the contents in said first register (W-ILC) and the contents in said second register (S-ILC);

a selecting means (34) for selecting the output of said first register (W-ILC) only when said second instruction does not have said special state, and for selecting the output of said adding means (33) only when said second instruction has said special state; and

a second adding means for adding the output of said selecting means (35) and the output of said program status word instruction address register (PSWIAR).

14. A pipeline operation control system as claimed in claim 12, said selecting means (34) has an inhibiting means for inhibiting the selection of the output of said adding means when an interrupt is detected in the execution of said first instruction.

15. A pipeline operation control system as claimed in claim 12, further comprising a program interrupt code register (PGMIC) for storing the cause of an interrupt.

16. A pipeline operation control system as claimed in claim 15, further comprising a first interrupt content storing register (WXC), a second interrupt content storing register (SXC), a register output selector (43) for selecting the contents in said first interrupt content storing register (WXC) or the contents in said second interrupt storing register (SXC) to be input to said program interrupt code register (PGMIC), said first interrupt content storing register (WXC) registering the contents of an interrupt of said first instruction when said first instruction has at least two calculating states immediately before the final state or registering the contents of an interrupt of said second instruction when said second instruction does not have said special state, and said second interrupt content storing register (SXC) registering the contents of an interrupt of said second instruction only when said second instruction has said special state, said register output selector (43) selecting the output of said second interrupt content storing register (WXC) unless said first interrupt content storing register (SXC) stores the contents of the interrupt of said second instruction.

Fig. 1

```
                                                    T
INSTRUCTION 1    D    A    T    B    E    W
     INSTRUCTION 2    D    A    T    B    E    W
          INSTRUCTION 3    D    A    T    B    E    W
               INSTRUCTION 4    D    A    T    B    E    W
                    INSTRUCTION 5    D    A    T    B    E    W
                         INSTRUCTION 6    D    A    T    B    E    W
```

Fig. 2

Fig. 3

INSTRUCTION 1	D	A	T	B	E	E	W		
INSTRUCTION 2		D	A	T	B	E	E	W	
INSTRUCTION 3			D	A	T	B	E	E	W
INSTRUCTION 4				D	A	T	B	E	E
INSTRUCTION 5					D	D	A	T	B
INSTRUCTION 6						D	D	A	T

Fig. 4

	D	A	T	B	E	V	W			
INSTRUCTION 1		D	A	T	B	E	V	W		
INSTRUCTION 2			D	A	T	B	E	V	W	
INSTRUCTION 3				D	A	T	B	E	V	W
INSTRUCTION 4|				D	A	T	B	E	V	W
INSTRUCTION 5|					D	A	T	B	E	V	W
INSTRUCTION 6|										

14

Fig. 5

INSTRUCTION 1	D	A	T	B	E	V	W				
INSTRUCTION 2		D	A	T	B	B	E	W			
INSTRUCTION 3			D	A	T	B	E	V	W		
INSTRUCTION 4				D	A	T	B	E	V	W	
INSTRUCTION 5					D	A	T	B	E	V	W
INSTRUCTION 6						D	A	T	B	E	V

Fig. 6A

Fig. 6

Fig. 6A	Fig. 6B

```
          START

                                          S61
    ┌──────────────────────────────────┐
    │ IN 1ST INSTRUCTION, DOES PROCE-  │  N
    │ SS ENTER INTO V STATES  ?        │────
    └──────────────────────────────────┘
                    Y
                                          S62
    ┌──────────────────────────────────┐
    │ CAN S STATES BE THE LAST         │  N
    │ STATES OF 2ND INSTRUCTION ?      │────
    └──────────────────────────────────┘
                    Y
                                          S63
    ┌──────────────────────────────────┐
    │ MAKE THE LAST STATE OF 2ND INSTRUCTION │
    │ TO BE SHIFTED FROM E STATE TO S STATE  │
    └──────────────────────────────────┘

    ┌──────────────────────────────────┐
    │ EXECUTE W STATE OF 1ST INSTRUCTION AND │
    │ S STATE OF 2ND INSTRUCTION AT THE SAME │
    │ TIME                                   │
    └──────────────────────────────────┘
                                          S64
                    S65
    ┌──────────────────────────────────┐
    │ INTERRUPT BECAUSE                │  N
    │ OF 1ST INSTRUCTION  ?            │────
    └──────────────────────────────────┘
                    Y    S66                    S67
    ┌──────────────────────────┐    ┌──────────────────────┐
    │ RETAIN INSTRUCTION COMPLETE │  │ RETAIN INSTRUCTION   │
    │ TAG FOR 1ST INSTRUCTION     │  │ COMPLETE TAG FOR     │
    │                             │  │ 2ND INSTRUCTION      │
    └──────────────────────────┘    └──────────────────────┘

    ┌──────────────────────────────────┐
    │ OBTAIN PSW1AR                    │  S68
    └──────────────────────────────────┘
```

16

Fig. 6B

```
┌──────────────────────────────┐
│  IN  2ND   INSTRUCTION        │
│  MAKE  LAST  TWO STATES       │──── S690
│  TO BE  V  AND  W             │
└──────────────────────────────┘

┌──────────────────────────────┐
│  EXCUTE  W  STATE  OF  IST    │
│  INSTRUCTION  AND  V  STATE   │──── S691
│  OF 2ND INSTRUCTION AT THE    │
│  SAME  TIME                   │
└──────────────────────────────┘

┌──────────────────────────────┐
│  OBTAIN   PSWIAR  OF          │──── S692
│  IST   INSTRUCTION            │
└──────────────────────────────┘

┌──────────────────────────────┐
│  EXECUTE  W  STATE  OF        │──── S693
│  2ND  INSTRUCTION             │
└──────────────────────────────┘

┌──────────────────────────────┐
│  OBTAIN   PSWIAR  OF          │──── S694
│  2ND  INSTRUCTION             │
└──────────────────────────────┘
```

Fig. 7

INSTRUCTION 1
INSTRUCTION 2 (S CYCLE INSTRUCTION)

Fig. 8

Fig. 9

STORE INSTRUCTION

B   E   W

GR
FR

13   10   CSB 21   STDR   22

Fig. 10

Fig. 11

Fig. 12

Fig. 13

```
        ┌─────────┐
        │  START  │
        └─────────┘
             │
            S71
         ╱────────╲
        ╱  IN 1ST  ╲
       ╱ INSTRUCTION,╲        N
      ╱  LAST TWO STATES ╲──────────────────────┐
      ╲  FROM V TO W     ╱                       │
       ╲  E→S (2nd)     ╱                       S76
        ╲    ?        ╱                      ╱────────╲
         ╲──────────╱                       ╱  E → W   ╲    N
             │ Y                            ╲    ?      ╱──────→
            S72                              ╲────────╱
  ┌──────────────────────────────┐              │ Y
  │ W−ILC ← V−ILC (1ST INST)     │             S77
  │ S−ILC ← E−ILC (2ND INST)     │    ┌──────────────────┐
  └──────────────────────────────┘    │ W−ILC ← E−ILC    │
             │                         └──────────────────┘
            S73                             │
  ┌──────────────────────┐                 S78
  │   S−ILC + W−ILC      │      ┌──────────────────────────┐
  └──────────────────────┘      │ PSWIAR ←PSWIAR + W−ILC   │
             │                  └──────────────────────────┘
            S74                             │
         ╱────────╲                     ┌─────────┐
        ╱ INTERRUPT╲      N             │  END    │
       ╱ BECAUSE OF 1ST╲───────┐        └─────────┘
       ╲ INSTRUCTION   ╱       │
        ╲    ?        ╱        │
         ╲──────────╱          │
             │ Y               │
            S75         ┌──────────────────────┐
  ┌──────────────────┐  │ PSWIAR ←             │
  │ PSWIAR ←PSWIAR   │  │ PSWIAR+S−ILC−W−ILC   │
  │   + W−ILC        │  └──────────────────────┘
  └──────────────────┘          │ S79
             │                   │
             │←──────────────────┘
        ┌─────────┐
        │  END    │
        └─────────┘
```

F l g. 14

Fig. 15

F l g. 16

EUROPEAN PATENT APPLICATION

⑫

㉒ Inventor : Asakawa, Takeo
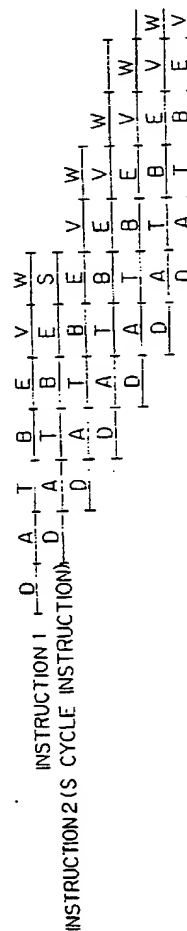c/o Fujitsu Limited, 1015, Kamikodanaka
Nakahara-ku, Kawasaki-shi, Kanagawa 211
(JP)
Inventor : Inoue, Aiichiro
c/o Fujitsu Limited, 1015, Kamikodanaka
Nakahara-ku, Kawasaki-shi, Kanagawa 211
(JP)

㊱ Representative : Billington, Lawrence Emlyn et
al
HASELTINE LAKE & CO Hazlitt House 28
Southampton Buildings Chancery Lane
London WC2A 1AT (GB)

�54 Pipeline operation control method and system.

�57 A pipeline operation control method and system in which, in the execution of continuous first and second instructions, to avoid a pipeline disturbance, the second instruction is shifted from a calculating state to a special state as a final state that does not conflict with the final state of the first instruction when the first instruction has two or more calculating states immediately before the final state, and when the special state can be inserted as the final state of the second instruction and the calculating state of the second instruction is carried out within a single machine cycle, whereby the final state of the first instruction and the special state of the second instruction are carried out at the same time.

Fig. 7

EP 0 503 968 A3

| | | European Patent Office | EUROPEAN SEARCH REPORT | | Application Number EP 92 30 2200 |

DOCUMENTS CONSIDERED TO BE RELEVANT

	Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.5)
3	A	1988 IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN: VLSI IN COMPUTERS & PROCESSORS October 1988 , NEW YORK pages 96 - 101 SETH ABRAHAM ET AL. 'Instruction Reorganization for a Variable-Length Pipelined Microprocessor' * the whole document *	1,2,12	G06F9/38
1	A	US-A-3 949 379 (INTERNATIONAL COMPUTERS LIMITED) 6 April 1976 * the whole document *	1,2,12	
4	A	EP-A-0 171 595 (INTERNATIONAL BUSINESS MACHINES, CORP.) 19 February 1986 * the whole document *	1,12	
1	A	EP-A-0 399 757 (TANDEM COMPUTERS INC.) 28 November 1990 * page 6, line 32 - page 7, line 8 * * page 8, line 18 - line 29 *	3,4,6, 8-10	
1	A	IBM TECHNICAL DISCLOSURE BULLETIN. vol. 32, no. 5A , October 1989 , NEW YORK US pages 33 - 36 'Roll-Back Interupt Method for Out-Of-Order Execution of System Programs' * the whole document *	11-13	TECHNICAL FIELDS SEARCHED (Int.Cl.5) G06F
1	A	IBM TECHNICAL DISCLOSURE BULLETIN. vol. 24, no. 11A , April 1982 , NEW YORK US page 5530 DALE ET AL. 'Pipeline Processor Exception Control Mechanism' * the whole document *	3-5,7-10	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	1 December 1993	Daskalakis, T